Out-of-Core Krylov-Subspace Methods

Sivan Toledo*
sivan@theory.lcs.mit.edu
MIT Laboratory for Computer Science

This abstract describes efficient out-of-core Krylov-subspace methods. Krylov-subspace methods include popular iterative linear solvers such as conjugate gradient and GMRES. If the data structures of the algorithm do not fit within main memory, then an out-of-core algorithm must be used. An out-of-core algorithms tries to be computationally efficient but in addition attempts to move as little data as possible between the computer's primary and secondary memories. An efficient conjugate-gradient algorithm described in this abstract is more than 3 times faster than naive implementations that rely on paging to perform I/O. This abstract summarizes results from [3].

Krylov-subspace methods are a class of iterative numerical methods for solving linear equations. Given an n-by-n matrix A and an n-vector b, Krylov-subspace methods compute a sequence of approximations $\{x^{(t)}\}$ to the solution x of the linear system of equations $Ax = b$, such that $x^{(t)} \in \text{span}\left\{b, Ab, A^2b, \ldots, A^tb\right\}$. The subspace $\mathcal{K}_t(A, b) = \text{span}\left\{b, Ab, A^2b, \ldots, A^tb\right\}$ is called a **Krylov** subspace. Krylov-subspace methods exist for eigenproblems as well.

Most Krylov-subspace methods involve several types of operations in every iteration, namely multiplication of a vector by the matrix A, vector operations (additions and multiplication by scalars), and inner products. Krylov-subspace algorithms that do not compute inner products are called **linear relaxation** algorithms. Almost all Krylov-subspace methods maintain a number of n-vectors in addition to $x^{(t)}$. Many reuse the space used to store these vectors, so the space required is only a small multiple of n words.

A naive approach to out-of-core algorithms is to use the in-core algorithm and let an automatic system, for example paging, transfer data on demand between levels of the memory hierarchy. The naive approach causes almost the entire data-structure that represents the state of the iterative algorithm to be read from secondary memory in every iteration. In many Krylov-subspace algorithms the performance of this naive approach is limited by the I/O channel that transfers data to and from secondary memory, because the amount of work per iteration is small.

We propose a novel implementation of Krylov-subspace algorithms, which is based on blockers [2]. Our method simulates τ iterations of a Krylov-subspace by performing τ iterations of linear relaxation twice, in addition to τ iterations of a Krylov-subspace algorithm on small data structures that fit within main memory. The advantage of this simulation technique is that the modified algorithm can use efficient out-of-core linear relaxation algorithms. Techniques for out-of-core linear relaxation are known for several classes of matrices, including matrices arising from relaxation on meshes and matrices arising from multigrid relaxation [2].

The Algorithm

The main idea behind our method is to block the flow of information through inner products in Krylov-subspace algorithms. We simulate τ iterations of the algorithm using three phases. In the first phase, we use linear relaxation to compute inner products, but we do not use them. Therefore, in Phase 1 information flows into the inner products but not out of them. In the second phase, we use the inner products from Phase 1 to compute the exact values of the inner products computed by the original Krylov-subspace algorithm. In the third phase we perform the Krylov-subspace algorithm. Since the values of the inner products are known, they are not computed, and hence, information flows out of these variables but not into them. we demonstrate our method on the following pseudocode that describes a simple Krylov-subspace algorithm for symmetric matrices A. Our method applies equally well to realistic algorithms such as conjugate gradient. Our method is related to algorithms by Chronopoulos and Gear [1] who proposed the first multistep Krylov-subspace algorithms.

SimpleKrylovMethod(A, x, τ)
for $t \leftarrow 1$ to τ
  do $y \leftarrow Ax$
    $\alpha_t \leftarrow x^{\mathrm{T}} x$
    $x \leftarrow x + \alpha_t y$

In Phase 1 we compute inner products without ever using them. Because inner products are not used, this phase is essentially a linear relaxation algorithm. We use an efficient out-of-core linear relaxation algorithm to perform this phase. The choice of algorithm depends on the structure of the matrix A. For some matrices a covering algorithm can be

* Sivan Toledo is advised by Charles E. Leiserson and is supported in part by ARPA contract N00014-94-1-0985. Address: Room 243, 545 Technology Square, Cambridge MA 02139. Web: http://theory.lcs.mit.edu/~sivan

n	CONJUGATEGRADIENT	CLUSTEREDDOTPRODSCG	KRYLOVBASISCG		
			$\tau = 5$	$\tau = 9$	$\tau = 15$
10K	23.2	22.8	60.7	58.9	58.7
50K	22.8	23.0	59.2	56.6	56.1
100K	25.9	197.5	77.1	67.0	61.0
250K	154.1	214.3	86.5	70.4	63.2
500K	245.8	211.4	85.6	70.3	62.8

Table 1 The running times of three implementations of the conjugate-gradient algorithm. The table shows the running time per element per iteration in microseconds. The second and third columns describe the running time of two naive implementations, whereas the last three columns describe the running time of our efficient out-of-core implementation. The experiments were performed on a Sun SPARCstation 1+ workstation with a Sun 4/60 CPU, 8Mbytes of main memory, and a local disk for paging. The programs were compiled by the GNU C compiler with a –02 optimization option. The numbers represent averages of three runs, each consisting of 10 phases of τ iterations. The running time per iteration of CONJUGATEGRADIENT and CLUSTEREDDOTPRODSCG is independent of τ, and experiments used $\tau = 5$. The matrix A is an n-by-n tridiagonal matrix which was partitioned into sets of at most 2000 elements.

used, and for other an algorithm based on blocking-covers is required [3]. Phase 1 can be easily modified so that only τ multiplications by A are required.

PHASEONE(A, x, τ)

$x_0 \leftarrow x$
for $t \leftarrow 1$ to $2\tau + 1$
  do $y \leftarrow Ax$
    $\beta_t \leftarrow x_0^{\mathrm{T}} x$
    $x \leftarrow y$

In Phase 2 we transform the vector β into the vector of α values required by the original algorithm. The full version of the abstract proves that the values produced are indeed the correct α's. The phase performs the Krylov algorithm in the Krylov basis $B = [x_0, Ax_0, \ldots, A^{\tau} x_0]$. It can be proved that in the original algorithm, x is always is in the column span of the Krylov matrix B, and therefore the algorithm can be expressed in this basis. We denote the representation of x in the Krylov basis by w, so that $x = Bw$. Therefore $x^{\mathrm{T}} x = w^{\mathrm{T}} B^{\mathrm{T}} B w$. We denote the $(\tau{+}1)$-by-$(\tau{+}1)$ matrix $B^{\mathrm{T}} B$ by Q. Multiplying a vector w by A in the Krylov basis is simple: the vector is shifted down by one. For moderate values of τ, this phase can be performed in core, since all its data structures are of size $\Theta(\tau)$. For larger numbers of iterations, we iterate all three phases several times.

PHASETWO(A, x, β, τ)

Form the matrix Q where $Q_{i,j} = \beta_{i+j-2}$
$w \leftarrow [1, 0, \ldots, 0]^{\mathrm{T}}$
for $t \leftarrow 1$ to τ
  do $v \leftarrow$ shift-down(w)
    $\alpha_t \leftarrow w^{\mathrm{T}} B^{\mathrm{T}} B w$
    $w \leftarrow w + \alpha_t v$

In Phase 3 we simple use the precomputed values of α. This phase too is a linear relaxation algorithm that can often be executed efficiently out of core.

PHASETHREE(A, x, α, τ)

for $t \leftarrow 1$ to τ
  do $y \leftarrow Ax$
    $x \leftarrow x + \alpha_t y$

Experimental Results

Table 1 shows that, On a workstation, our Krylov-basis algorithm with $\tau = 9$ outperforms the original conjugate-gradient algorithm by up to a factor of 3.5, when data structures do not fit within main memory. Our algorithm outperforms the CLUSTEREDDOTPRODSCG algorithm by a factor of up to 3. When the data fits within main memory, our algorithm is about 2.6 times slower than the other two, because it performs more work. But by performing more work and less I/O, our algorithm is faster on large problems. The Krylov-basis algorithm is significantly faster then the other implementations even with $\tau = 5$.

References

[1] A. T. Chronopoulos and C. W. Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25:153–168, 1989.

[2] Charles E. Leiserson, Satish Rao, and Sivan Toledo. Efficient out-of-core algorithms for linear relaxation using blocking covers. In *Proceedings of the 34rd Annual Symposium on Foundations of Computer Science*, pages 704–713, November 1993. To appear in the *Journal of Computer and System Sciences*.

[3] Sivan A. Toledo. *Quantitative Performance Modeling of Scientific Computations and Creating Locality in Numerical Algorithms*. PhD thesis, Massachusetts Institute of Technology, 1995.